International Journal of Engineering & Technology, 7 (3.29) (2018) 670-673

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

A Study on the Operating Systems for Low End IOT Devices

Anusha R[1*], B.Soundarya[2], G.Priyanka[3], V.Soniya [4]

[1] *Assistant Professor, St.Peters Engineering College,Hyderabad*
**Corresponding Author E-mail : amanuknr@gmail.com*

Abstract

The IOT interconnect billions of devices and they can be connected to internet. IOT run on high end devices which use traditional operating systems like Linux and low end devices with less computational power, memory etc. The large scale deployment of the system will need appropriate OS for the development, deployment and maintenance of the devices. In this paper we research on OS that could run on low end devices for IOT.

Keywords: *IOT operating systems, Low end devices, Contiki, RIOT, Free RTOS*

1. Introduction

IOT emerged with the availability of cheap, energy efficienttiny devices. Many protocols were developed for IOT network stack which enabled these devices reachable from internet. The heterogeneous devices used in IoT are of two types. The first category consists of high end devices like smart phones, Raspberry Pi, etc. The second category uses low end devices like the Arduino, Zolertia Z1, OpenMote nodes. These highly constrained low end devices are challenging for the OS designers.
Internet Engineering Task Force (IETF) classified these low end devices into 3 based on their memory capacity.
Class 0 devices with the smallest resources (<<10 kB of RAM)
Class 1 devices have medium-level resources (~10 kB of RAM)-allows applications with advance features- E.g. routing and secure communication protocol
Class 2 devices have more resources but constrained compared to high end devices
The Class 0 device's, resource constraints leads to using a proper OS unsuitable. Thus, the software running on them are hardware-specific. Class 2 devices are less specialized The software can transform these devices into routers, servers or host with standard network stack and reprogrammable applications running on the stack. Thus new models emerge on these portable hardware independent software and applications running on class 1 IOT devices and above. Companies like Google, Huawei, ARM have designed OS specifically for these IOT devices. Application programming interfaces (APIs) will support the wide range of IoT use cases, to facilitate large-scale software development, deployment, and maintenance which can be provide by the OS. In this paper we will focus on OS for class 1 and class 2.
The traditional OS like Linux can't run on low end IOT devices because of resource constraint. Thus IOT faces lack of interoperability with many solutions. Thus only with the emergence of OS with consistent API and SDK across IOT platform will they be able to use their potential. The interoperability of IOT devices with internet means compatibility with network IP protocols, compatibility with programming languages, tools used in internet

2. Requirement For An IOT OS

Here we give an overview on the requirement of low end IOT device OS

2.1. Technical Properties

1. Small memory: The IOT devices are resource constrained specially in terms of memory. These devices can provide only Kilobytes of memory. Thus the OS for IOT devices must fit into these memory constraints. It must provide the IOT application designers a set of optimized library and efficient data structures. Balance must be found between programming guidelines, coding conventions and configurability to fit wide range of use cases.
2. Heterogeneous Hardware support: IOY devices work with 8 bit, 16 bit 32 bit microcontroller families. The Heterogeneity of devices occur as many types of devices work together for a particular task. Thus the OS for IOT devices must support this Heterogeneity of hardware and communication technologies.
3. Network connectivity: The main feature of IoT is interconnectivity. The devices in Iot are interconnected among themselves and to internet. Thus they are equipped with network interfaces. Communication technologies are used to address low power radio technologies, and wired technologies. Thus the IoT OS must support this combination of multiple link technology and network stacks based on IP.
4. Energy efficiency: Most of the IoT devices run on batteries or similar energy resources. The IoT hardware has features to run in an energy efficient manner. Hence unless the IoT software utilizes these features energy efficiency can't be achieved. Thus the IoT OS must provide energy saving options and use these functions as much as possible
5. Real Time Capabilities: Precise timing and timely execution is must for IoT devices. The IoT OS must be a real time OS which can fulfill timely execution and operates with a deterministic run time.

Copyright © 2018 Authors. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

6. Security: The IoT device are connected to internet and are must meet security and privacy standards. IoT security challenges include data integrity, authentication, and access control in various parts of the IoT architecture. Thus, the OS for the IoT must provide the necessary mechanisms and retain the flexibility and usability

2.2. Nontechnical Properties

1. Open standards: An important feature of OS is to provide portability across architectures without any effort. Standard APIs allow software portability among OS. At the network level, the open-access specification, by the IETF is preferred. These standards improve portability and interoperability.
2. Certification: The critical system developed in electrical/electronic safety related systems requires real-time capabilities, robustness, or determinism. Thus it is unavoidable to get certified through independent institutions like IEC 61508 standard. The entire software running on IoT system must be certified when used for safety critical systems.
3. Documentation: Any software requires complete and easily understandable documentation. The documentation for embedded systems must be intense and easily understood as it they are difficult to understand in the first sight. The complete and understandable documentation simplifies the use of OS and application design.
4. Licence and Maturity of code: There are 3 types of licences- non free which is given as binary code or they will charge to get the source code. The second type is permissive open source which is highly accepted in industry due to their high degree of freedom. The third type which is open source or copyleft licenses offers higher code quality and secure code due to extensive contributors and reviewers. The maturity of the OS can be measured if the OS is widely deployed in applications after thorough testing.
5. OS provider: The OS code may be given by the OS developer or by a third party who provides commercial support. The open source OS will be distributed by the developer themselves which will provide support through online forums. The way of distribution and supportof OS is dependent on its licence.

2.3. Key Design Choices

The applicability of Os for IoT depends on technical and organizational factors. Here we will review the technical and non technical aspects of OS design.
1. Architecture and modularity: When any OS is designed the kernel model is first to be thought of. The exokernel approach focus on avoiding resource conflict and checking access level. Microkernel approach provides more functionality in the kernel with less memory and more flexibility. The monolithic approach, the components of the system are developed together which leads to more efficient and simpler design.
2. Scheduling model: There are 2 types of scheduler. The pre-emptive scheduler which can interrupt the current task and give control to another after a particular time limit. The non pre-emptive or cooperative model doesn’t allow the interruption of any running task. The pre-emptive scheduler needs a timer to be active at any given time which prevents the IoT device to enter into deepest power save model
3. Memory Allocation: As memory is a scarce resource in IoT devices, sophisticated methods to handle them is required. The 2 methods af memory allocation are static method and dynamic method. Static methods don’t provide flexibility during runtime. To use dynamic memory allocation for applications with real-time requirements, the OS has to provide special implementations for deterministic malloc() and must find solutions to deal with out-of –memory situations. It leads to a more complex system and may conflict with real-time requirements.
4. Network buffer management: The central component of IoT OS is the network stack which allows memory sharing among the layers. The memory handling may be achieved by copying the memory which is expensive or by passing pointers between layers which is not flexible and convenient. Thus a central memory manager must be used to resolve the memory for packet handling either by allocating them to each level or by passing reference between layers.
5. Programming model: This defines how programmer can model the program. There are 2 ways by which we can get it done. In event driven method every task must be triggered by an external event which is achieved by an event loop. In multithreaded model the developer can run each task in its own thread and communicate using IPC API. The event driven systems are memory efficient and multithreading makes application design easier.
6. Programming Languages and debugging tools: The programming languages used can be either standard programming languages like C,C++ or OS specific languages. The use of standard programming languages will enhance portability and can use well known development tools. They also enable the use of standard debugging tools. The OS specific languages improve system performance and safety.
7. Driver model and hardware abstraction layer: IoT systems must interact with the environment passively through sensors and actively through actuators. Many peripheral devices are also connected to them. Thus a flexible and convenient driver interface is important for IoT OS. This can abstract the underlying hardware from CPU, memory etc. This can improve the system design even though it creates the runtime over head.
8. Testing: The distributed nature and constraints of the hardware makes thorough testing a challenging, but crucial task. Hardware related testing is done using hardware emulation tools or networks emulators like MSPSim, Emul8 etc

3. Categories Of OS:

1. Event driven OS: This is the most promising approach for IOT OS. This model all the events are triggered by external interrupts. This approach is efficient in memory utilization and low complexity. IT poses some substantial constraint for programmer as all programs can’t be represented in finite state machine.
2. Multithreading: This is a traditional approach for most of the OS used in high end devices. In this approach each thread run its own work and manages its own stack. It produces some memory overhead due to stack over provisioning and context switching.
3. Pure RTOS: This provides more real time features for industrial and commercial context. The strict constraints of this model makes it inflexible and difficult to port to other hardware.

4. Promising Iot OS

In this section we will discuss about some of the promising IoT OS. Here we will discuss about open source, closed source OS. Most of the OS is written in C language while some hardware specific part may be implemented in Assembly level languages.

4.1. Open Source OS

1. Contiki[2][3]: Contiki was developed by Adam Dunkels in 2002 and is the most widely used OS for constrained nodes. Initially it was used for memory constrained 8 bit MCUs used in WSN. Now it’s used for 32 bit ARM processors. It has a monolithic kernel with a core system and set of process acting together as a single system. It supports event driven, co-operative scheduling with light weight pseudo threading. It also supports various network stacks like uIP stack which supports IPv6, 6LoWPAN, RPL, and CoAP and Rime Stack which supports distributed programming abstractions. It also provides features of a shell, a file system, a database management system, runtime dynamic linking, cryptography libraries, and a fine grained power tracing tool. The testing facilities include unit testing, regression testing, and full system integration testing. Its code can be automatically tested using TravisCI.

Table I: Overview of OS for IoT

Name	Architecture	Scheduling	Programming Model	Targeted Device Class	Supported MCU Families	Programming Languages	Licence	Network Stack
Contiki	monolithic	Cooperative	Event driven	Class 0+1	AVR, ARM 7, ARM cortex-M	C	BCD	uIP, RIME
RIOT	Microkernel	Preemptive, tickless	Multithreading	Class 1+2	AVR, ARM 7, ARM cortex-M, x86	C, C++	LGPLv2	Open WSN, ccn-lite
Free RTOS	Microkernel	Preemptive	Multithreading	Class 1+2	AVR, ARM, x86, Renasas	C	Modified GPL	none

2. RIOT [4]–[6]: RIOT was developed as a developer friendly programming model and API in 2013. It is a micro-kernel based RTOS along with multithreading. Special efforts are put to develop efficient context switching, IPC (blocking and non blocking), and a small thread control block to deal runtime overhead. The tickless scheduler will put RIOT into deepest sleep mode which will be woken by interrupts. RIOT supports network stacks, including its own 6LoWPAN stack, 6TiSCH stack, Open WSN, CCN-lite. The default network stack gncr stores the metadata in a centralized network buffer and communication by passing pointers between layers. It has a wide range of features, such as a shell, crypto libraries, or sophisticated data structures. RIOT provides a set of unit tests and applications for smoke and regression testing. CI testing is performed on the web-based service platform Travis.

3. Free RTOS [7]: It was developed by Richard Barry in 2002 with a pre-emptive microkernel which also supports multithreading. It has a simple architecture with functionalities like thread handling mutexes, semaphores, and software timers. It doesn't have its own network stack but supports the 3rd party network stacks. Real Time Engineers Ltd. offers an official Free RTOS +TCP add-on supporting an Ethernet-based IPv4 stacks with support for UDP, TCP, and supporting protocols. Testing and debugging is also performed by 3rd party solutions.

4.2. Closed Source OS:

There are several closed OS preferable for IoT devices. Some vendors offer limited access to their source code for the users. These OS are typically designed for other domains and lack features like energy saving mechanisms and standardized protocols. Below are few examples of OS which run on class 0 and class 1 devices.

Table II: Key features of OS for IoT

Name	Category	MCU/ MMU	<32KB RAM	6LoWPAN	RTOS scheduler	HAL	Energy Efficient MAC Layer
Contiki	Event driven	Yes	Yes	Yes	No	Yes	Yes
RIOT	Multithreading	Yes	Yes	Yes	Yes	Yes	No
free RTOS	RTOS	Yes	Yes	No	Yes	No	No

1.ThreadX [8]: This is developed by Express Logic and taken over by ARM. It is a microkernel RTOS with multithreading and pre-emptive scheduler. Network stack, file system, GUI must be purchased as separate products.

2. VxWorks [9]: developed by Wind River in 1983. It has a monolithic kernel that supports ARM and Intel platforms. It supports IPv6 but not 6LoWPAN stack.

3. PikeOS [10]: This was developed by SYSGO AG in 1991.This is a microkernel RTOS which supports safety and security and also acts as hypervisor for many OS. It has multiple API and also supports guest OS.

5. Conclusion:

In this paper we have analyzed the various requirements for an OS that could be used for IoT low end devices. We have surveyed the available OS which could be used best. We have focused on open source OS code which provides higher transparency, trust worthiness and security. WE have also identifies 3 categories of OS which are equivalent to Linux. In multithreaded OS RIOT is the most prominent OS. Event driven OS are designed to fit devices with less resources of which Contiki is the most prominent open source OS. RTOS focus on worst case execution time and interrupt latency in which Free RTOS is the best. In the long run, the nature of most open source OS increases the probability and fits better the needs of SMEs. According to recent studies such companies will
be driving IoT innovation in the near future. According to our study we conclude that there are many OS for IoT for users to choose from depending on their requirements. As the IoT field is developing at a rapid pace, however, the final word is yet to be made regarding what type of architecture and capabilities an ideal OS for the IoT should have.

References

[1] A.Dunkels, B.Grönvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in Proc. 29th Annu. Int. Conf. Local Comput. Netw. (LCN), 2004, pp. 455–462 [Online].Available: http://dblp.uni-trier .de /db/ conf/lcn /lcn2004.html# DunkelsGV04

[2] "Contiki operating system," [Online]. Available: http://www.contikios.org.

[3] E.Baccelli,O.Hahm,M.Günes,M.Wählisch,andT.C.Schmidt,"RIOT OS: Towards an OS for the Internet of Things," in Proc. 32nd IEEE INFOCOM, 2013.

[4] E. Baccelli, O. Hahm, H. Petersen, and K. Schleiser, "RIOT and the evolution of IoT operating systems and applications," ERCIM News, vol. 2015, no. 101, 2015 [Online]. Available: http://ercim-news.ercim. eu/en101/special/riot-and-the-evolution-of-iot-operating-systems-and applications

[5] "RIOT operating system," [Online]. Available: http://www.riot-os.org

[6] "Berkeley's OpenWSN project," [Online]. Available: http://openwsn. berkeley.edu/

[7] R. Barry. "FreeRTOS, a free open source RTOS for small embedded real time systems," [Online]. Available; http://www.freertos.org

[8] Express Logic Inc. "ThreadX," [Online]. Available: http://rtos.com/ products/threadx/

[9] Wind River Syst. "VxWorks," [Online]. Available: http://www.windriver. com/products/vxworks/

[10] SYSGO.“PikeOS,”[Online].Available:http://www.sysgo.com/products/ pikeos-rtos-and-virtualization-concept/
[11] E. Upton and G. Halfacree, Meet the Raspberry Pi. Hoboken, NJ, USA: Wiley, 2012.
[12] Arduino due,” [Online]. Available: http://arduino.cc/en/Main/ arduinoBoardDue
[13] Redwire Llc. “Redwire Econotag II,” [Online]. Available:http://redwire.myshopify.com/products/econotag-ii
[14] Oliver Hahm, Emmanuel Baccelli, Hauke Petersen, and Nicolas Tsiftes, Operating Systems for Low-End Devices in the Internet of Things: A Survey, IEEE Internet Of Things Journal, Vol. 3, No. 5, October 2016